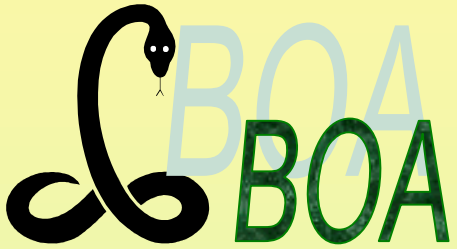


Framework for Automated Builds

Natalia Ratnikova

CHEP'03

Project Goals

- Facilitate software maintenance, help to improve software quality in the areas of software development, release management and distribution processes with the aids of the automated builds.
- Systematize available tools and components and put them all to work together in a highly automated fashion.
- Provide knowledge base management system for the automated software builds.

System Requirements

- Set up tools to collect code tags and automatically build the corresponding software on the range of supported CMS platforms.
- Hyperlinked log files for builds should be provided.
- The system should also support building of pre-releases, releases and even private test builds.

Analyzing Existing Systems

- Nightly ATLAS software releases on Linux platform
  http://www.usatlas.bnl.gov/computing/software/nightlies/www/nightly_builds.html
- Night builds of ALICE off-line software
  http://alisoft.cern.ch/offline/aliroot-pro/nightbuilds.html
- CDF run II software management
  http://cdfkits.fnal.gov/
- Continuous integration with Cruisecontrol
  Http://cruisecontrol.sourceforge.Net/
- GCC testing efforts
  http://gcc.gnu.org/testing/
- *… and many others*

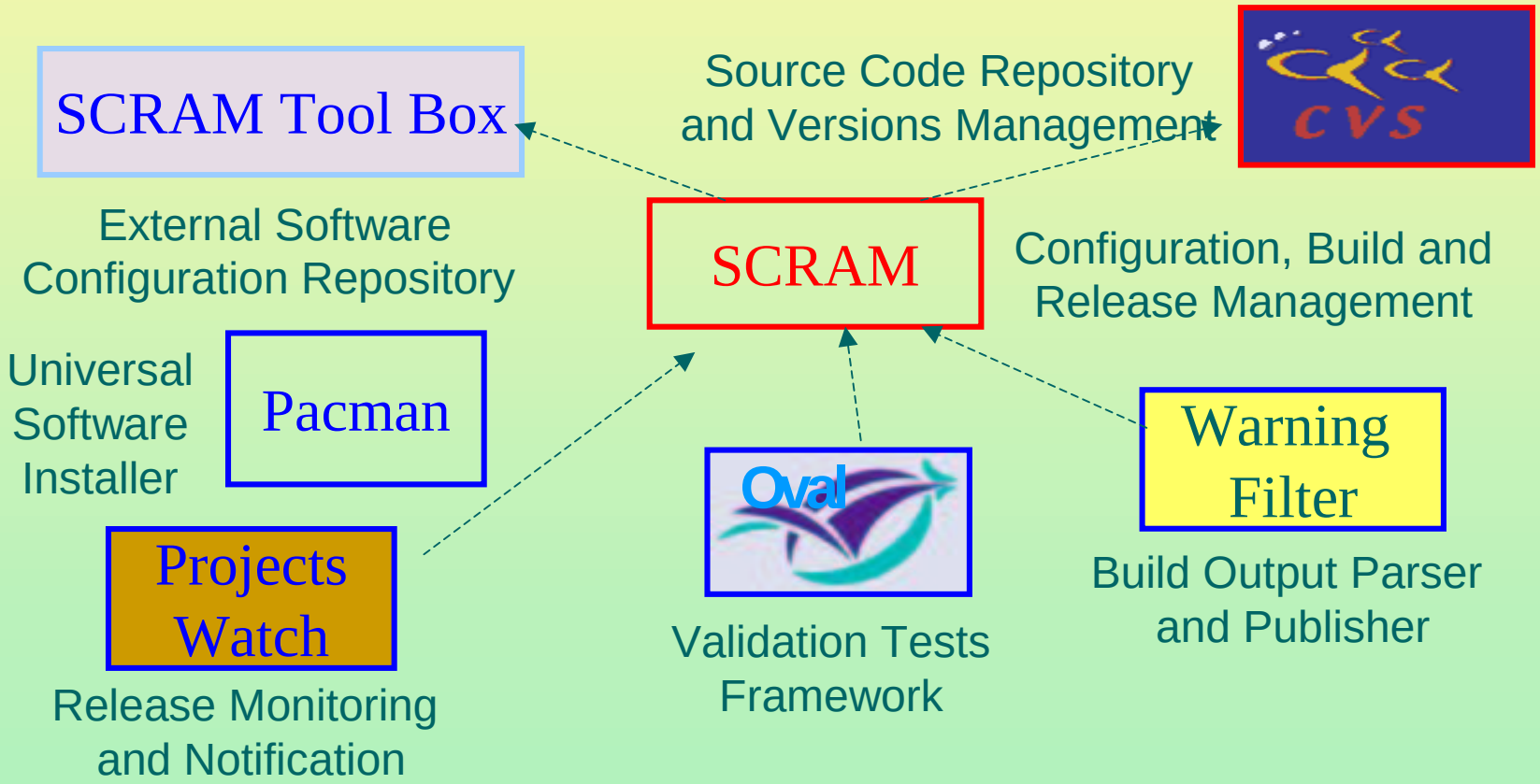
CMS Case Study

- Massive software development.
  - Over 300 public software releases for the last 15 months.
  - World-wide distributed development.
  - Multiple software projects with anisochronous release schedule.
  - Cross dependencies between projects.
  - External products and tools.
  - Multiple platforms
  - compiler and OS upgrade

Statistics to be added…

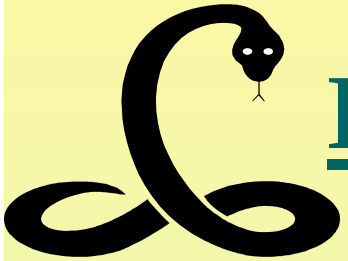
CMS Case Study(cont)

- Versatile supporting tools and services are already available in the CMS environment:



CMS Case Study

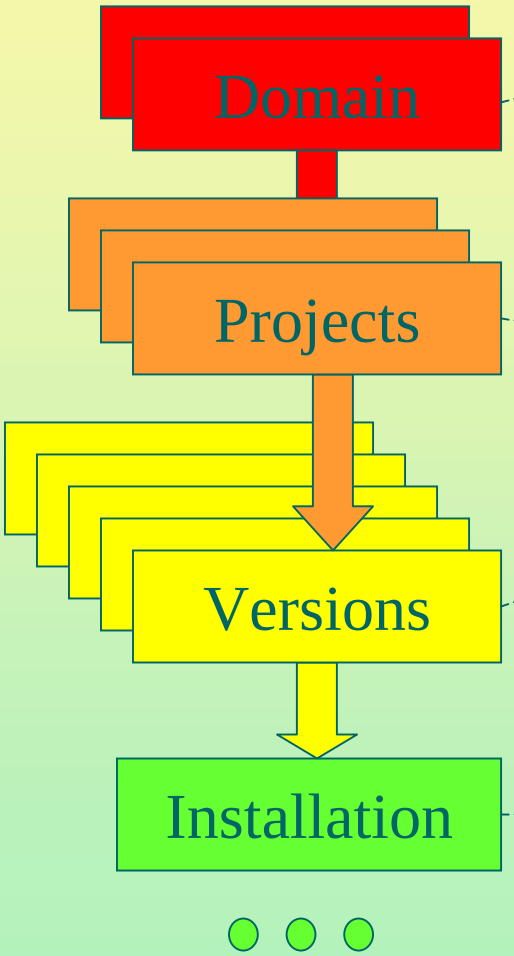
- Main challenge: Configuration
  - Cross-projects dependencies.
  - Plenty of detailed specific information.
  - Changing requirements and infrastructure.
  - Lack of standardization for regular operations.
  - Complicated and dynamic dependencies between components.
  - Site specific configuration management (most tedious and error-prone operations).



BOA Solutions:

- MODEL:
  - Look for invariants in the complex system
  - Abstract the structure from the functionality, and the functionality from the implementation
- IMPLEMENTATION:
  - Accumulate and systematize the knowledge base
  - Provide standard interfaces to inter-changeable components
- PROCESS:
  - Cyclic development and early prototyping
  - Constantly testing and documenting

System Architecture



Maintains list of projects, site dependent information, defines projects types, provides access to projects user interface

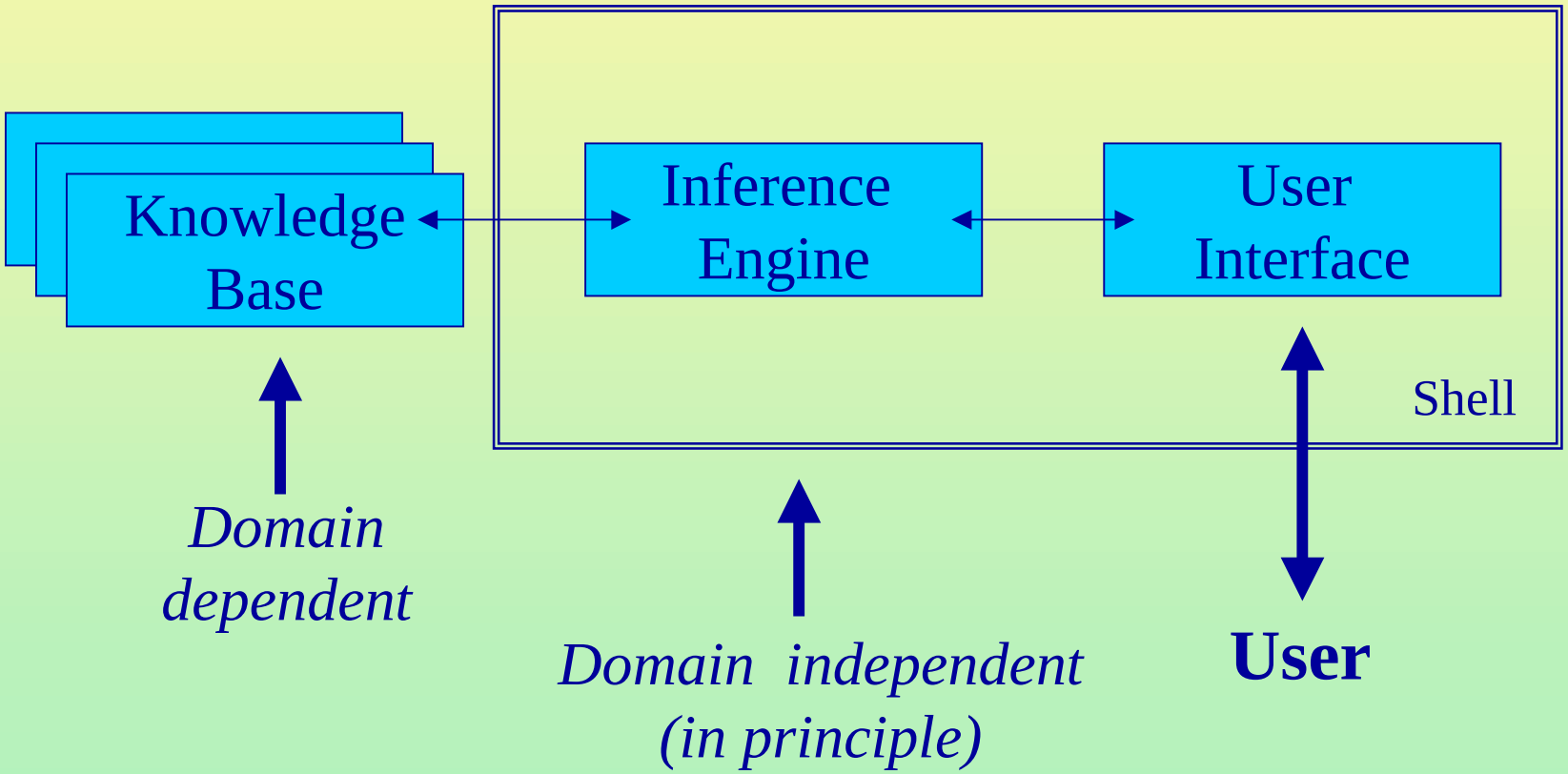
Maintains list of versions.
Implements type specific operations.

Maintains different types of installation (e.g. **Release**, **Nightly**, **Integration**…).
Contains configuration requirements.

Handles installation specific information and algorithms

BOA Framework Architecture

Corresponds to basic expert system architecture



System Features

- Proposed architecture allows to separate services from the implementation details
  - supports multiple types of projects: *scramified, pacmanized*… and builds: *releases, nightlies*,…
  - allows for customized configuration on the domain level
  - provides convenient user interface with built-in help and standard interfaces to the underlying pluggable components.
  - domain database keeps track of multiple projects, versions, installations and their status

Implementation and Status

- First experience with Perl-written prototype while providing solid base for algorithmic part of the system, did not allow required flexibility in configuration.
- Object oriented model has been developed, base BOA classes and their responsibilities have been identified and implemented in Python.
- Base classes: Framework, Domain, Project, Version, Installation, Platform,… and their major subclasses are implemented.

Implementation and Status (Cont)

- Utilities module provides support for:
  - Users command line interface with built-in help (based on standard python module cmd)
  - Child-parent components architecture
  - Persistency mechanisms (pickle)
  - Abstract factory for support multiple types of components
  - Logger with built-in timer
- Work on the algorithmic part and standard interfaces to the components is currently in progress.